The Rosetta Stone
Making COCOMO 81 Estimates Work with COCOMO II

Donald J. Reifer, *Reifer Consultants, Inc.*
Barry W. Boehm and Sunita Chulani, *University of Southern California*

As part of our efforts to help Constructive Cost Model (COCOMO) users, we, the COCOMO research team at the Center for Software Engineering at the University of Southern California (USC), have developed the Rosetta Stone to convert COCOMO 81 files to run using the new COCOMO II software cost estimating model. The Rosetta Stone is extremely important because it allows users to update estimates made with the earlier version of the model so that they can take full advantage of the many new features incorporated into the COCOMO II package. This article describes both the Rosetta Stone and guidelines to make the job of conversion easy.

During the past few years, the COCOMO team at USC has been working to update the 1981 version of the COCOMO estimating model (COCOMO 81) [1]. The new version of the model, called COCOMO II, builds on the experiences that industrial affiliates of the USC Center for Software Engineering have had and addresses lifecycle processes and paradigms that have become popular since the original model was first introduced in 1981. These new paradigms include reuse-driven approaches, commercial-off-the-shelf lifecycle developments, component-based software engineering approaches, use of object-oriented methods, and other improvements to the way we do business stimulated by process improvement initiatives.

This article focuses attention on a tool we have developed to permit our users to update their original COCOMO 81 files so that they can be used with the COCOMO II model. We call the tool the Rosetta Stone because it is not unlike the black stone slab found in Egypt by French troops in 1799. On it were engraved three scripts (Greek, Demotic, and hieroglyphics), which enabled archaeologists to construct translations among the three languages. Our Rosetta Stone permits its users to translate files prepared with the original COCOMO 81 model to be compatible with COCOMO II.

Many of our affiliates thought creation of the Rosetta Stone was important because they had wanted to use the new version of the model to take advantage of its many advanced capabilities, including the COCOMO II package's autocalibration features. However, they could not make the move because they had files that required older versions of the model to run, e.g., COCOMO 81 and Ada-COCOMO. Others wanted to calibrate the new version of the model using their historical databases, but the new version of the model had a new structure, altered mathematics, and different parameters and parametric ratings. Under such circumstances, converting files was no easy task.

The COCOMO II Estimating Model

The major differences between COCOMO 81 and COCOMO II, why they are important, and cost driver definitions are summarized in Table 1. These changes are important because they reflect how the state of software engineering technology has matured during the past two decades. For example, programmers were submitting batch jobs when the COCOMO 81 model was first published. Turnaround time impacted their productivity. Therefore, a parameter TURN was used in the model to reflect the average wait programmers experienced before receiving their job back. Such a parameter is no longer important because most programmers have instant access to computational facilities through their workstations. Therefore, the parameter has been removed in the COCOMO II model.

The following summary highlights the major changes made to the original version of COCOMO 81 as COCOMO II was developed.

- COCOMO II addresses the following three phases of the spiral lifecycle: applications development, early design, and post-architecture.
- The three modes in the exponent are replaced by five scale factors.
- The following cost drivers were added to COCOMO II: DOCU, RUSE, PVOL, PEXP, LTEX, PCON, and SITE.
- The following cost drivers were deleted from the original COCOMO: VIRT, TURN, VEXP, LEXP, and MODP.
- The ratings for those cost drivers retained in COCOMO II were altered considerably to reflect more up-to-date calibrations.

The Rosetta Stone

As illustrated in Table 2, users need to convert factors in the COCOMO equations (such as the exponent, the size estimate, and the ratings for the cost drivers) from the original to the new version of the model. We suggest that users employ the following four steps to make the conversion so original files can be used with the COCOMO II model.

Update Size

The original COCOMO cost estimating model used deliverable source lines of

	COCOMO 81	COCOMO II
Model Structure	Single model that assumes you start with requirements allocated to software.	Three models that assume you progress through a spiral-type development to solidify your requirements, solidify the architecture, and reduce risk.
Mathematical Form of Effort Equation	Effort = $A(c_i)(Size)^{Exponent}$	Effort = $A(c_i)(Size)^{Exponent}$
Exponent	Exponent = Fixed constant selected as a function of mode. • Organic = 1.05 • Semidetached = 1.12 • Embedded = 1.20	Exponent = Variable established based on rating of five scale factors. • PREC: Precedentedness • FLEX: Development Flexibility • RESL: Architecture or Risk Resolution • TEAM: Team Cohesion • PMAT: Process Maturity
Size	SLOC (with extensions for FPs).	Object points, FPs, or SLOCs.
Cost Drivers (c_i)	15 drivers, each of which must be rated • RELY: Reliability • DATA: Database Size • CPLX: Complexity • TIME: Execution Time Constraint • STOR: Main Storage Constraint • VIRT: Virtual Machine Volatility • TURN: Turnaround Time • ACAP: Analyst Capability • PCAP: Programmer Capability • AEXP: Applications Experience • VEXP: Virtual Machine Experience • LEXP: Language Experience • TOOL: Use of Software Tools • MODP: Use of Modern Programming Techniques • SCED: Required Schedule	17 drivers, each of which must be rated • RELY: Reliability • DATA: Database Size • CPLX: Complexity • RUSE: Required Reusability • DOCU: Documentation • TIME: Execution Time Constraint • STOR: Main Storage Constraint • PVOL: Platform Volatility • ACAP: Analyst Capability • PCAP: Programmer Capability • AEXP: Applications Experience • PEXP: Platform Experience • LTEX: Language and Tool Experience • PCON: Personnel Continuity • TOOL: Use of Software Tools • SITE: Multisite Development • SCED: Required Schedule
Other Model Differences	Model based on • Linear reuse formula. • Assumption of reasonably stable requirements.	Has many other enhancements, including • Nonlinear reuse formula. • Reuse model that looks at effort needed to understand and assimilate. • Breakage ratings used to address requirements volatility. • Autocalibration features.

Table 1. *Model comparisons.*

code (DSI) as its measure of the size of the software job. DSI were originally represented by card images, e.g., includes all non-comment, nonblank carriage returns. COCOMO II uses the following three measures to bound the volume of work associated with a software job: source lines of code (SLOC), function points (FPs), and object points. SLOCs are counted using logical language statements per Software Engineering Institute (SEI) guidelines [2], e.g., `IF-THEN-ELSE, ELSE IF` is considered one, not two, statements.

Table 2 provides guidelines to convert size in DSI to SLOCs so that they can be used with the COCOMO II model. Whenever possible, we recommend using counts for the actual size of the file instead of the original estimate. Such practices allow you to correlate your actuals, e.g., the actual application size with the effort required to do the work associated with developing the software.

The reduction in size for COCOMO II estimates is attributable to COCOMO 81's need to convert card images to SLOC counts. As already noted, the pair `IF-THEN-ELSE` and `END IF` would be counted as two card images in COCOMO 81 and as a single source instruction in COCOMO II. The guidelines offered in Table 2 are based on statistical averages to simplify conversions; however, we encourage you to use your actuals if you have them.

The following are two common misconceptions about COCOMO's use of SLOC and FPs:

- *Misconception 1: COCOMO does not support the use of FPs* – FP versions of COCOMO have been available since the Before You Leap commercial COCOMO software package implementation in 1987. As noted in Table 1, COCOMO II supports use of either SLOC or FP metrics. In both cases, this is done via "backfiring" tables, which permit you to convert FPs to SLOCs at different levels.
- *Misconception 2: Although it is irresponsible to use SLOC as a general productivity metric, it is not irresponsible to use FP as a general sizing parameter for estimation* – This misconception breaks down into the two following cases.
  - Your organization uses different language levels to develop software. In this case, it is irresponsible to use SLOC as your productivity metric, since you get higher productivity per SLOC at higher language levels; however, it also is irresponsible to use FP as a general sizing metric because pure FP will generate the same cost (or schedule or quality) estimate for a program with the same functionality developed

COCOMO 81	COCOMO II
DSI • Second-Generation Languages • Third-Generation Languages • Fourth-Generation Languages • Object-oriented Languages	SLOC [3] • Reduce DSI by 35 percent. • Reduce DSI by 25 percent. • Reduce DSI by 40 percent. • Reduce DSI by 30 percent.
Function Points	Use the expansion factors developed by Capers Jones [4] to determine equivalent SLOCs.
Feature Points	Use the expansion factors developed by Capers Jones to determine equivalent SLOCs.

Table 2. *Converting size estimates.*

using different language levels. This is clearly wrong. To get responsible results in this case, FP-based estimation models need to use some form of backfiring to account for the difference in language level.

- Your organization always uses the same programming language (level). Here, it is responsible to use pure FP as your sizing metric for estimation. But it also is responsible to use SLOC as your productivity metric. Both metrics work in practice.

Convert Exponent

Convert the original COCOMO 81 modes to scale factor settings using the Rosetta Stone values in Table 3. Then, adjust the ratings to reflect the actual situation. For example, the Rosetta Stone rates process maturity (PMAT) low because most projects using COCOMO 81 are assumed to have been at Level 1 on the SEI process maturity scale [5]. However, the project's actual rating may have been higher and an adjustment may be in order.

An exception is the PMAT scale factor, which replaces the COCOMO 81 Modern Programming Practices (MODP) multiplicative cost driver. As seen in Table 4, MODP ratings of very low (VL) or low (L) translate into a PMAT rating of VL or a low level on the Software Capability Maturity Model scale. An MODP rating of normal (N) translates into a PMAT rating of L or a high Level 1. An MODP rating of high (H) or very high (VH) translates into a PMAT rating of N or CMM Level 2. As with the other factors, if you know that the project's actual rating was different from the one provided by the Rosetta Stone, use the actual value.

Table 3. *Model scale factor conversion ratings.*

Mode and Scale Factors	Organic	Semidetached	Embedded
Precedentedness (PREC)	XH	H	L
Development Flexibility (FLEX)	XH	H	L
Architecture and Risk Resolution (RESL)	XH	H	L
Team Cohesion (TEAM)	XH	VH	N
Process Maturity (PMAT)	L	L	L

The movement from modes to scale factors represents a major change in the model. To determine the economies and diseconomies of scale, five factors have been introduced. Because each of these factors can influence the power to which size is raised in the COCOMO equation, they can have a profound impact on cost and productivity. For example, to increase the rating from H to VH in these parameters can introduce as much as a 6 percent swing in the resulting resource estimate. Most of these factors are modern in their derivation. For example, the concept of process maturity was not in its formative stages when the original COCOMO 81 model was published. In addition, the final three factors, RESL, TEAM, and PMAT, show how an organization can exercise management control over its diseconomies of scale. Finally, the first two, PREC and FLEX, are the less controllable factors contributing to COCOMO 81 modes or interactions.

Rate Cost Drivers

The trickiest part of the conversion is the cost drivers. Cost drivers are parameters to which cost is sensitive. For example, as with the scale factors, you would expect that use of experienced staff would make a software development less expensive; otherwise, why use them? Because the new version of the model uses altered drivers, the Rosetta Stone conversion guidelines outlined in Table 4 are important. For those interested in more details about the cost drivers, we suggest you refer to the COCOMO II Model Definition Manual [6]. Again, the ratings need to be adjusted to reflect what actually happened on the project. For example, the original estimate may have assumed that analyst capability was very high; however, the caliber of analysts actually assigned might have been nominal because key employees were not available to the project when they were needed.

Users should take advantage of their knowledge of what occurred on the project to make their estimates more reflective of what really went on as the application was developed. Use of such knowledge can improve the credibility and accuracy of their estimates.

The TURN and TOOL rating scales have been affected by technology changes since 1981. Today, virtually everyone uses interactive workstations to develop software. TURN has therefore been dropped from COCOMO II and its calibration assumes the TURN rating is L. Table 5 provides alternative multipliers for other COCOMO 81 TURN ratings.

The tool suites available in the 1990s far exceed the COCOMO 81 VH TOOL rating, and virtually no projects operate at the COCOMO 81 VL or L TOOL levels. COCOMO II has shifted the TOOL rating scale two levels higher so that a COCOMO 81 N TOOL rating corresponds to a VL COCOMO II TOOL rating. Figure 5 also provides a

COCOMO 81 Drivers	COCOMO II Drivers	Conversion Factors
RELY	RELY	None. Rate the same or the actual.
DATA	DATA	None. Rate the same or the actual.
CPLX	CPLX	None. Rate the same or the actual.
TIME	TIME	None. Rate the same or the actual.
STOR	STOR	None. Rate the same or the actual.
VIRT	PVOL	None. Rate the same or the actual.
TURN		Use values in Table 5.
ACAP	ACAP	None. Rate the same or the actual.
PCAP	PCAP	None. Rate the same or the actual.
VEXP	PEXP	None. Rate the same or the actual.
AEXP	AEXP	None. Rate the same or the actual.
LEXP	LTEX	None. Rate the same or the actual.
TOOL	TOOL	Use values in Table 5.
MODP	Adjust PMAT settings.	If MODP is rated • VL or L, set PMAT to VL. • N, set PMAT to L. • H or VH, set PMAT to N.
SCED	SCED	None. Rate the same or the actual.
	RUSE	Set to N or actual, if available.
	DOCU	If Mode: = Organic, set to L. = Semidetached, set to N. = Embedded, set to H.
	PCON	Set to N or actual, if available.
	SITE	Set to H or actual, if available.

Table 4. *Cost drivers conversions.*

set of COCOMO II multipliers corresponding to COCOMO 81 project ratings.

Some implementations of COCOMO II, such as the USC COCOMO II package, provide slots for extra user-defined cost drivers. The values in Figure 5 can be put into those slots (if you do this, use an N rating in the normal COCOMO II TOOL slot).

To learn more about the cost drivers and their ratings, refer to the USC Web site (http://sunset.usc.edu/COCOMOII) or several of the Center for Software Engineering's other publications [7, 8]. Because the goal of this article is to present the Rosetta Stone, we did not think it was necessary to go into the details of the model and an explanation of its many parameters.

Experimental Accuracy

To assess the accuracy of the translations, the team used the Rosetta Stone to convert 89 projects. These projects were clustered subsets of the databases we used for model calibration. Clusters were domain-specific. We updated our estimates using actuals whenever we could. We then used the autocalibration feature of the USC COCOMO II package to develop a constant for the effort equation, e.g., the A in the equation $\text{Effort} = A(\text{size})^P$. Finally, we compared our estimates to actuals and computed the relative error as a function of the following cases.

- Using the Rosetta Stone with no adjustments.
- Using the Rosetta Stone with knowledge-base adjustments, e.g., updating the estimate files with actuals when available.
- Using the Rosetta Stone with knowledge-base adjustments and domain clustering, e.g., segmenting the data based on organization or application area.

The results of these analyses, which are summarized in Table 6, were extremely positive. They show that we can achieve an acceptable degree of estimating accuracy when using the Rosetta Stone to convert COCOMO 81 files to run with the COCOMO II software cost model.

Summary and Conclusions

The Rosetta Stone was developed to provide its users with a process and a tool to convert their original COCOMO 81 files so that they can be used with the new COCOMO II estimating model. The Stone represents a starting point for such efforts. It does not replace the need to understand either the scope of the estimate or the changes that occurred as the

Table 5. *TURN and TOOL adjustments.*

COCOMO 81 Rating	VL	L	N	H	VH
COCOMO II Multiplier: TURN		1.00	1.15	1.23	1.32
COCOMO II Multiplier: TOOL			1.24	1.10	1.00

Table 6. *Estimate accuracy analysis results.*

Cases	Accuracy (Relative Error)
Using the COCOMO II model as calibrated.	Estimates are within 25 percent of actuals 68 percent of the time.
Using the COCOMO II model as calibrated using developer or domain clustering.	Estimates are within 25 percent of actuals 76 percent of the time.
Using the Rosetta Stone with no adjustments.	Estimates are within 25 percent of actuals 60 percent of the time.
Using the Rosetta Stone with knowledge-base adjustments.	Estimates are within 25 percent of actuals 68 percent of the time.
Using the Rosetta Stone with knowledge-base adjustments and domain clustering.	Estimates are within 25 percent of actuals 74 percent of the time.

project unfolded. Rather, the Stone takes these considerations into account as you update its knowledge base with actuals.

The value of the Rosetta Stone was demonstrated convincingly based on an accuracy analysis of an 89-project database. As expected, the accuracy increased as we adjusted the estimates using actuals and looked at results based on domain segmentations. We are encouraged by the results. We plan to continue our efforts to provide structure and support for such conversion efforts. ◆

References

1. Boehm, B., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
2. Park, R., "Software Size Measurement: A Framework for Counting Source Statements," CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh, Pa., 1992.
3. Reifer, D., personal correspondence, 1998.
4. Jones, C., *Applied Software Measurement: Assuring Productivity and Quality*, McGraw-Hill, New York, 1992.
5. Paulk, M., C. Weber, B. Curtis, and M. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading, Mass., 1995.
6. Boehm, B., et al., *COCOMO II Model Definition Manual, Version 1.4*, University of Southern California, Los Angeles, Calif., 1997.
7. Boehm, B., et al., "The COCOMO 2.0 Software Cost Estimation Model," *American Programmer*, July 1996, pp. 2-17.
8. Clark, B. and D. Reifer, "The Rosetta Stone: Making Your COCOMO Estimates Work with COCOMO II," *Software Technology Conference*, Salt Lake City, Utah, 1998.

About the Authors

Donald J. Reifer is a leading figure in software engineering and management, with over 30 years progressive experience in government and industry. He has been chief of the Ada Joint Program Office, technical adviser to the Center for Software, and director of the Department of Defense (DoD) Software Reuse Initiative under an Intergovernmental Personnel Act assignment with the Defense Information Systems Agency. He is currently president of RCI, a small consulting firm servicing Fortune 500 companies, and he is a visiting associate at USC, where he serves on the COCOMO team. He has a bachelor's degree in electrical engineering, a master's degree in operations research, and a certificate in business management (master's equivalent). His many honors include the Secretary of Defense's medal for Outstanding Public Service, the NASA Distinguished Service Medal, the Freiman Award, and the Hughes Aircraft Fellowship. He has over 100 publications, including his popular *IEEE Software Management Tutorial* and a new Wiley book entitled *Practical Software Reuse*.

Reifer Consultants, Inc.
P.O. Box 4046
Torrance, CA 90505
Voice: 310-530-4493
E-mail: d.reifer@ieee.org

Barry W. Boehm is considered one of the fathers of the field of software engineering. He is currently director of the Center for Software Engineering at USC, and for many years directed key technology offices in the DoD, TRW, and Rand Corporation. His contributions to software engineering include COCOMO, the spiral model of the software process, the Theory W approach to software management and requirements determination, and the TRW Software Productivity System and Quantum Leap advanced software engineering environments. His current software research interests include process modeling, requirements engineering, architectures, metrics and cost models, engineering environments, and knowledge-based engineering.

Boehm has a bachelor's degree from Harvard University and a master's degree and a doctorate from the University of California at Los Angeles, all in mathematics. He has served on the board of several scientific journals and has served as chairman of numerous prominent engineering society committees. He is the recipient of many of software engineering's highest awards. He is an American Institute of Aeronautics and Astronautics Fellow, an Association for Computing Machinery Fellow, an Institute of Electrical and Electronics Engineers Fellow, and a member of the National Academy of Engineering.

Center for Software Engineering
University of Southern California
941 West 37th Place
Los Angeles, CA 90089
Voice: 213-740-8163
E-mail: boehm@sunset.usc.edu

Sunita Chulani is a research assistant at the Center for Software Engineering at the University of Southern California. She is an active participant on the COCOMO II research team and is working on a Bayesian approach to data analysis and model calibration. She is also working on a cost and quality model that will be an extension to the existing COCOMO II model. Her main interests include software process improvement with statistical process control, software reliability modeling, risk assessment, software cost estimation, and software metrics. She has a bachelor's degree in computer engineering from Bombay University and a master's degree in computer science from USC. She is currently a doctoral candidate at the Center for Software Engineering at USC.

Center for Software Engineering
University of Southern California
941 West 37th Place
Los Angeles, CA 90089
Voice: 213-740-6470
E-mail: sdevnani@sunset.usc.edu